manuscript No.(will be inserted by the editor)

A Versatile Hardware/Software Platform for Personalized Driver Assistance based on Online-Sequential Extreme Learning Machines

Inés del Campo · Victoria Martínez · Javier Echanobe · Estibalitz Asua · Raúl Finker · Koldo Basterretxea

Received: date / Accepted: date

Abstract In the present scenario of technological breakthroughs in the automotive industry, machine learning is greatly contributing to the development of safer and more comfortable vehicles. In particular, personalization of the driving experience using machine learning is an innovative trend that comprises the development of both customized driver assistance systems (DAS) and in-cabin comfort features. In this work, a versatile hardware/software platform for personalized driver assistance, using online sequential extreme learning machines (OS-ELM), is presented. The system, based on a programmable system-on-chip (SoC), is able to recognize the driver and personalize the behavior of the car. The platform provides high speed, small size, efficient power consumption, and true capability for real-time adaptation (i.e. on-chip self-learning). In addition, due to the plasticity and scalability of the OS-ELM algorithm and the programmable nature of the SoC, this solution is flexible enough to cope with the incremental changes that the new generation of vehicles are demanding. The implementation details of a system, suitable for current levels of driving automation, are provided.

I. del Campo
E-mail: ines.delcampo@ehu.eus

I. del Campo, V. Martínez, J. Echanobe, E. Asua, and R.Finker
Department of Electricity and Electronics, Faculty of Sciences and Technology, University of the Basque Country (UPV/EHU), 48940 Leioa, Spain

Present address: of R. Finker
CELESTIA S.L.U., c/ Albert Einstein 14, 39011 Santander, Spain

K. Basterretxea
Department of Electronics Technology, EUITI, University of the Basque Country (UPV/EHU), 48013 Bilbao, Spain

Keywords Driver Assistance Systems (DAS) · Extreme Learning Machine · Online learning · Multiobjective Optimization · Field Programmable Gate Arrays (FPGA) · System-on-chip (SoC)

1 Introduction

New cars are equipped with a wide variety of sensors, actuators, and electronic control units (ECU), all of them interacting in the background with the aim of improving the experience of driving and travelling by car. The major challenge that automotive industry has to face over the next few years is concerned with the continuous improvement of driver assistance systems (DAS), until reliable self-driving systems for all traffic situations and driving modes become available [1]. In addition, as the driver input in performing the driving task decreases, vehicles are turning into safe spaces where people will spend part of their time. Therefore, enhancing the comfort and well-being of people traveling in them is also an ongoing challenge [2].

In the above scenario, personalization of vehicles is recognized as an important research focus [1,2]. Adapting vehicles to the users preferences and requirements will contribute to improve safety, security and comfort of drivers and passengers. A key task in vehicle personalization is the identification of the driver. Different driver identification techniques have been successfully tested in cars, ranging from simple personal keys or smartphones, to biometric features such as face scan [3], voice processing [4], or less intrusive features such as the sitting posture [5] or a set of driving behavior signals [6,7,8,9,10,11]. Recently, a driver identification methodology using historical trip-based data collected through an user-identified device (e.g. smartphone), to serve as complement to existing in-vehicle data recorder technologies, is proposed [12]. However, only simple methods, easily hacked, are currently available in commercial vehicles.

Concerning the driving task, the aim of a personalized DAS is to mimic the natural behaviour of a driver in manually driving a vehicle. As a result, the response of DAS is more transparent and intuitive for the user, improving both perceived and actual driving safety and comfort [13]. Recently, several experiments performed with personalized adaptive cruise control (ACC) and automatic lane change systems concluded that personalized solutions enhance driver acceptance and trust in automated functionalities [14, 15]. Besides, concerning personalization of in-cabin comfort, a variety of electronically configurable features are being developed to suit users preferences and needs: air conditioning, lightening, seats position, mirrors and steering wheel position, or infotainment settings, among others. Currently the driver can adjust these features using control buttons, a touch-screen or voice commands, depending on vehicle equipment, while a number of commercial cars incorporate memory functions for memorizing individual settings of a driver, or a reduced group of drivers. The desired car configuration can be retrieved at any time on driver request, or automatically if the car is equipped with a driver identification system [16, 17].

Machine learning techniques have proven useful in modelling many of the above-mentioned features. A number of techniques, mainly artificial neural networks (ANN), based on both shallow and deep architectures, are increasingly present in the automotive sector [18, 19]. However, most applications based on these techniques can only be dealt with using offline learning, meanwhile in-vehicle self-learning technology (i.e. autonomous online learning) is still in its infancy. In this sense, a high-speed machine learning method, known as extreme learning machine (ELM), has emerged as a suitable solution for online learning in cutting edge applications [20, 21, 22]. The most remarkable features of the ELM learning algorithm are: i) it is based on a simple tuning-free algorithm, ii) learning with ELM does not present local minima or over-fitting problems, iii) and its learning speed is very high. In addition, the online sequential implementation of ELM (OS-ELM) is able to deal with the training data as they are read (i.e. in real-time) [23]. As a consequence, ELM is less dependent on designer intervention than conventional machine learning techniques, such as back-propagation (BP) ANN, or support vector machines (SVM). All these characteristics make ELM suitable for the development of autonomous single-chip DAS and in-car integration.

Currently there is a rapid expansion of commercial system-on-chip (SoC) offered for implementing DAS [24]. Several companies offer hardware platforms consisting of a mixture of general purpose processors, digital signal processors (DSP), application-specific hardware accelerators, memory blocks, and a set of peripherals [25, 26, 27, 28]. These kinds of approaches achieve very high performance and integration, but at the expense of an underutilization of resources and processing power in the deployment of particular DAS applications. To deal with this problem, field programmable gate array (FPGA) technology provides the means to achieve a trade-off solution to the implementation of DAS [29]. These platforms are able to conciliate the need for application-specific energy-efficient processors and the need for more on-chip system integration to achieve higher levels of efficiency and reliability at a lower cost. Moreover, the suitability of FPGAs for the development of fault-tolerant ANNs is demonstrated in [30, 31]. Fault tolerance of embedded electronic systems is becoming increasingly important in the automotive sector where safety-critical applications are involved.

In this work, a versatile hardware/software (HW/SW) platform for personalized driver assistance using a programmable SoC is presented. The proposed architecture integrates general purpose microprocessors (SW partition) plus a configurable and scalable set of specialized cores based on an ELM paradigm to efficiently run data processing at high speed (HW partition). The SW partition performs system monitoring and control, input/output (I/O) processing, and machine self-learning, while the set of OS-ELM cores implemented in the HW partition is in charge of the personal assistance of the driver in real-time.

In sum, the personalization of the whole driving experience using a versatile platform with self-learning capability can be highlighted as an innovative contribution of the proposal. In addition, due to the plasticity of the ELM paradigm, able to deal with both classification and regression problems, a diversity of in-vehicle applications can be developed and implemented in the platform. Regarding the efficiency of the platform, implementing exclusively the required cores, tailored for each DAS application, and integrating only the required peripherals avoids resource underutilization and unnecessary power consumption. Finally, the platform is flexible enough to cope with the incremental changes that the new generation of vehicles is demanding. It is able to evolve and to adapt to the requirements of each level of automation, without introducing significant architectural changes, reducing thus costs and time-to-market, while enhancing vehicle safety and reliability.

The rest of the paper is organized as follows. First, in Section 2, the basics of OS-ELM are reviewed. In addition, to deal with the trade-off complexity/performance of ELM, a multi-objective optimization method is introduced and the whole proposed methodology is summarized. Section 3 describes the programmable SoC-based architecture and provides resource usage and achievable operation frequencies. In Section 4 three kinds of personalized applications are implemented using the platform: a security feature for anti-theft impostor detection, a safety application for ACC driving assistance, and an air recirculation feature as an in-cabin comfort application.

2 Optimized Extreme Learning Machine

ELM consists of a single-hidden layer feed-forward network (SLFN) endowed with a very efficient learning algorithm that reduces human interaction. In particular, the online sequential ELM (OS-ELM) is able to handle data one-by-one or on a chunk-(block of data)-by-chunk scheme, making it suitable for training dynamic systems that are receiving data in real-time. The advantages of ELM-like algorithms arise from the fact that the parameters of the hidden nodes are randomly generated and do not need to be iteratively tuned. As a consequence, the learning procedure of ELM is simpler and less time-consuming than conventional learning algorithms such as BP-ANN. This distinctive characteristic of ELM is especially important when on-chip adaptation is required. However, on the other hand, several experiments performed with real-world applications showed that an adequate selection of the number of hidden nodes is critical to achieve good generalization ability [32]. To avoid this drawback, a multi-objective optimization technique will be proposed with the aim of optimizing the neural network topology of medium/large-size applications.

2.1 Extreme Learning Machine

First, the basics of ELM are briefly reviewed with the aim of highlighting the advantages of this machine learning technique and providing the required background [21]. Figure 1 depicts the topology of a SLFN with n inputs, m outputs, and L nodes in the hidden layer. The network output for generalized batch ELM with additive nodes is

$$y(\mathbf{x}) = \sum_{i=1}^{L} \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta}, \tag{1}$$

without loss of generalization, a single output node ($m = 1$) is taken in (1). The vector of weights $\boldsymbol{\beta} = [\beta_1, \cdots, \beta_L]$ links the hidden nodes (i.e. random nodes) with the output node, and $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \cdots, h_L(\mathbf{x})]$ is the output vector of the hidden layer for a given input $\mathbf{x} \in \mathbf{R}^n$. The output of the ith hidden node is

$$h_i(\mathbf{x}) = S(\mathbf{a}_i, b_i, \mathbf{x}) = s(\mathbf{a}_i\mathbf{x} + b_i), \mathbf{a}_i, \mathbf{x} \in \mathbf{R}^n, \mathbf{b}_i \in \mathbf{R}, \tag{2}$$

with $s(\mathbf{a}_i\mathbf{x} + b_i)$ being the sigmoid activation function, $\mathbf{a}_i$ the random weight vector connecting the inputs with the ith hidden node, and b_i the random bias of the ith hidden node.

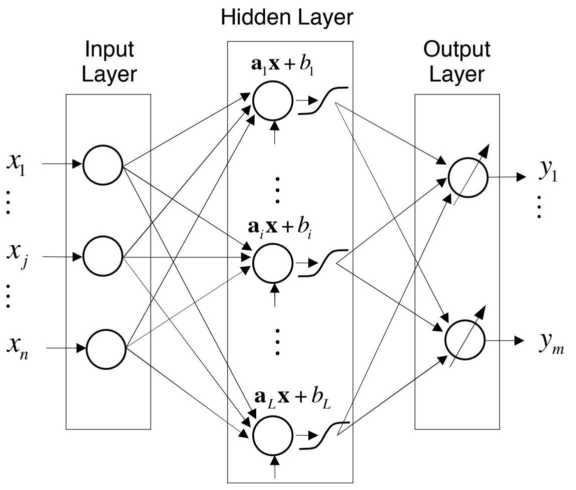

Hidden Layer
Input Layer
Output Layer
$\mathbf{a}_1\mathbf{x} + b_1$
$\mathbf{a}_i\mathbf{x} + b_i$
$\mathbf{a}_L\mathbf{x} + b_L$
x_1
x_j
x_n
y_1
y_m

Fig. 1 Topology of a single-layer feed-forward network (SLFN) used by ELM. The weights and biases of the hidden layer are random numbers, while the parameters of the output layer are analytically determined.

2.1.1 Batch Learning with ELM

The set of parameters of the hidden nodes $(\mathbf{a}_i, b_i)$, $1 \leq i \leq L$, are randomly generated and they are not tuned. This is important in an embedded solution because a set of random numbers can be generated out of the chip and further stored in a read-only memory (ROM).

Learning in batch ELM aims at computing the vector of output weights, $\boldsymbol{\beta}$ in Eq. (1), for each output node. Given a set of K training samples, $(\mathbf{x}_j, \mathbf{t}_j)$, with $1 \leq j \leq K$, where $\mathbf{x}_j \in \mathbf{R}^n$ is the jth input vector, and $\mathbf{t} \in \mathbf{R}^m$ is the corresponding output vector (i.e. the target output), learning is performed by solving Eq. (1) for the set of training samples

$$\mathbf{T} = \mathbf{H}(\mathbf{x})\mathbf{B}, \tag{3}$$

with $\mathbf{H}$ being the hidden layer output matrix

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_K) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_K) & \cdots & h_L(\mathbf{x}_K) \end{bmatrix}_{K\times L} \tag{4}$$

$$\mathbf{B} = \begin{bmatrix} \boldsymbol{\beta}_1 \ldots \boldsymbol{\beta}_m \end{bmatrix}_{L\times m}, \text{ and } \mathbf{T} = \begin{bmatrix} \mathbf{t}_1 \\ \vdots \\ \mathbf{t}_K \end{bmatrix}_{K\times m} \tag{5}$$

Then, Eq. (3) is a linear system and the output weights $\mathbf{B}$ can be estimated as

$$\hat{\mathbf{B}} = \mathbf{H}^{\dagger}\mathbf{T}, \tag{6}$$

where $\mathbf{H}^{\dagger}$ is the Moore Penrose generalized inverse of matrix $\mathbf{H}$. Different methods can be used to solve Eq. (6), with the singular value decomposition (SVD) method being the most used with ELM [33].

2.1.2 Learning with OS-ELM

The above batch ELM algorithm assumes that the whole set of training samples is available to train the system. This method is useful when historical data are provided and the system dynamics does not change over time. However, in most real-time applications, the training samples are read one-by-one or chunk-by-chunk, and changes in the system over time are expected, mainly when human behavior is involved (e.g. driver behavior). The batch ELM algorithm can be modified to allow online sequential training [23]. When the rank of matrix $\mathbf{H}$ equals the number of hidden nodes, $rank(\mathbf{H}) = L$, then $\mathbf{H}^{\dagger}$ in Eq. (6) can be computed by means of the left pseudoinverse of $\mathbf{H}$

$$\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T, \tag{7}$$

Substituting Eq. (7) into Eq. (6), an alternative solution of Eq. (3), known as least-squares solution, is obtained

$$\hat{\mathbf{B}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T}, \tag{8}$$

The online sequential implementation of Eq. (8) is the basis of OS-ELM. It is worth noting that OS-ELM and ELM are able to achieve the same learning performance whenever the initialization training set (i.e. the initial number of observations) verifies that $rank(\mathbf{H}_0) = L$. This means that the number of samples in the first chunk K_0 should be equal or greater than the number of hidden nodes L.

Learning with OS-ELM involves two main steps: the initialization phase, and the sequential learning phase. In the initialization phase the number of hidden nodes L is defined and the first chunk of data $C_0 = \left\{(\mathbf{x}_j, \mathbf{t}_j)\right\}_{j=1}^{K_0}$ is used to train the network, where $K_0 \geq L$. Then, the matrix of output weights that minimizes $\left\|\mathbf{H}_0\mathbf{B} - \mathbf{T}_0\right\|$ can be computed by means of Eq. (8)

$$\mathbf{B}^{(0)} = (\mathbf{H}_0^T\mathbf{H}_0)^{-1}\mathbf{H}_0^T\mathbf{T}_0, \tag{9}$$

After that, the process enters into the sequential learning phase. Each time a new chunk $C_{k+1} = \left\{(\mathbf{x}_j, \mathbf{t}_j)\right\}_{j=K_0+K_1+\ldots+K_k+1}^{K_0+K_1+\ldots+K_k+K_{k+1}}$ of K_{k+1} data is read, the new set of output weights $\mathbf{B}^{(k+1)}$ is computed as a function of the previous weights $\mathbf{B}^{(k)}$, and the pair of partial matrices $\mathbf{H}_{k+1}$ and $\mathbf{T}_{k+1}$. It is worth mentioning that only the samples belonging to the new chunk of data are used to obtain $\mathbf{H}_{k+1}$ and $\mathbf{T}_{k+1}$, while the initial chunk of data can be forgotten [23],

$$\begin{aligned}\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} + (\mathbf{H}_k^T\mathbf{H}_k + \mathbf{H}_{k+1}^T\mathbf{H}_{k+1})^{-1} \\ \mathbf{H}_{k+1}^T(\mathbf{T}_{k+1} - \mathbf{H}_{k+1}\mathbf{B}^k).\end{aligned} \tag{10}$$

2.2 Multi-objective optimization of ELM topology

There is a trade-off between the complexity of the system (i.e. the size of the SLFN depicted in Fig. 1) and the system performance when a hardware implementation is developed. To deal with this problem, a multi-objective optimization strategy, based on a genetic algorithm, is proposed.

Let $\mathbf{f}(z)$ be a p-dimensional vector of functions to be minimized, $\mathbf{f}(z) = (f_1(z), f_2(z), \ldots, f_p(z))$, the goal in a multi-objective optimization problem (MOP) is to find values of z that simultaneously minimizes the p functions $f_i(z), i = 1, \ldots, p$. However, if any of the functions compete with each other (e.g. ELM performance and size of the SLFN), there is no unique solution to the problem. Instead, the optimization process provides a set of trade-off solutions, called Pareto-optimal, in which at each point an improvement in one objective requires a degradation of another [34]. The final decision rests on the designer who has to select the most suitable solution on the basis of his or her experience and the system requirements.

Consider the optimization problem of an ELM system. The MOP can be stated as a 3-dimensional vector of functions to be minimized $\mathbf{f}(z) = (f_1(z), f_2(z), f_3(z))$, with the first and the second function being concerned with the system complexity and the third with the system performance:

$$\mathbf{f}(z) = \begin{cases} f_1(z) = \sum_{j=1}^{f} z_j\text{: input features } (n) \\ f_2(z) = z_{f+1}\text{: hidden neurons } (L) \\ f_3(z) = \text{ testing error of the SLFN,} \end{cases} \tag{11}$$

with $z \in \mathbf{Z}$ being a decision variable composed of f binary values and a single natural value: $\mathbf{Z} = \{0,1\}^f \bigcup \mathbf{N}$. The binary values are related with the available variables or features in such a way that the jth feature is selected (i.e. it becomes an input to the neural network) if $z_j = 1$, and is rejected if $z_j = 0$. The optimization algorithm searches in the $(f+1)$-dimensional space the values of $z \in \mathbf{Z}$ that minimize $\mathbf{f}(z)$.

While the search is being performed, the algorithm must evaluate the candidate solutions, where each evaluation involves training ELM with the selected features and hidden nodes, and testing the corresponding performance.

Therefore, a systematic combinatorial search would be in practice unfeasible, even for a fast algorithm such as ELM, due to the large amount of possible solutions. To tackle this problem, a multi-objective optimization genetic algorithm (MOGA) is applied. In particular, the well-known NSGA-II (non-dominate sorting genetic algorithm II) will be used [35]. The individuals of the population (i.e. chromosomes) to be evolved by the genetic algorithm are different SLFN topologies in the space of solutions **Z**. This kind of algorithm provides a set of Pareto-optimal solutions in a reasonable time. The MOGA is intended for off-line selection of suitable SLFN topologies, therefore, its implementation is not critical neither in time nor in resources (i.e. a personal computer can be used to run the algorithm).

2.3 Methodology Summary and Design Parameters

The most relevant aspect of the proposed methodology is the suitability of ELM and OS-ELM for the development of personalized DAS. This assertion relies on two distinctive characteristics of ELM-based solutions: autonomy (i.e. low human intervention is required) and real-time training capability. These features are difficult to achieve with conventional machine learning techniques, such as BP-ANNs or SVM solutions, mainly because of the strong dependency of these paradigms on their design parameters. Figure 2 summarizes the main steps involved in the design and on-line operation of an OS-ELM application: the SLFN topology design, the on-line sequential learning steps, and the system processing in real-time.

First, during the off-line design step, the estimation of the design parameters is performed according to Eq. (11). The pareto-optimal solutions used are aimed at selecting a suitable number of features (n) and hidden neurons (L), while the number of outputs (m) is determined by the application itself. These parameters describe the topology of the neural network, namely *n-L-m* SLFN. In addition, during the off-line step, a set of random numbers are pre-computed and stored in a ROM memory with the aim of using them as random weights and bias. The off-line design step can also cover a batch ELM learning procedure, not shown in Fig. 2, whenever the training samples are available. Batch ELM models can be used as initial models for further real-time evolution of the output weights using new chunks of data.

The online-step is intended for OS-ELM training using chunks of data as they are read. The size of the chunks (K) depends on the availability of observations and the timing requirements of each application. During the initialization phase, the output weights are computed using the first chunk of data. Then, the sequential learning phase is activated: a new chunk of data is read and the new set of output weights is computed. The OS-ELM training procedure finishes when all the chunks of data have been processed. The training procedure can be activated again, with the aim of updating the learning machine performance at any time. The set of output weights are then transferred to the OS-ELM core in the hardware partition where they are stored in a RAM memory. Finally, the real-time processing of I/O samples can be activated. The details of the OS-ELM core implementation, as well as the whole system architecture, are given in the following section.

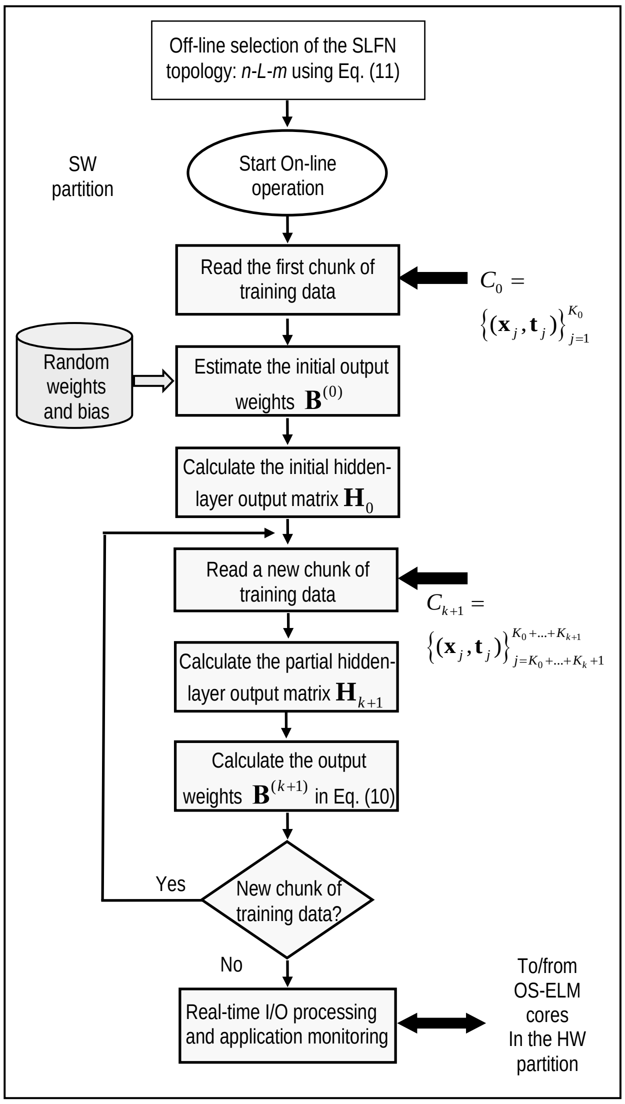

Off-line selection of the SLFN topology: n-L-m using Eq. (11)
SW partition
Start On-line operation
Read the first chunk of training data
$C_0 = \{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{K_0}$
Random weights and bias
Estimate the initial output weights $\mathbf{B}^{(0)}$
Calculate the initial hidden-layer output matrix $\mathbf{H}_0$
Read a new chunk of training data
$C_{k+1} = \{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=K_0+...+K_k+1}^{K_0+...+K_{k+1}}$
Calculate the partial hidden-layer output matrix $\mathbf{H}_{k+1}$
Calculate the output weights $\mathbf{B}^{(k+1)}$ in Eq. (10)
Yes
New chunk of training data?
No
Real-time I/O processing and application monitoring
To/from OS-ELM cores In the HW partition

Fig. 2 Flow chart with the steps involved in the topology design (off-line step) and the real-time system training and operation (on-line step).

3 SOC Architecture for the Personalized Platform

Figure 3 depicts a block diagram of the HW/SW platform for personalized driver assistance developed in this work. The architecture of the system is intended for a current

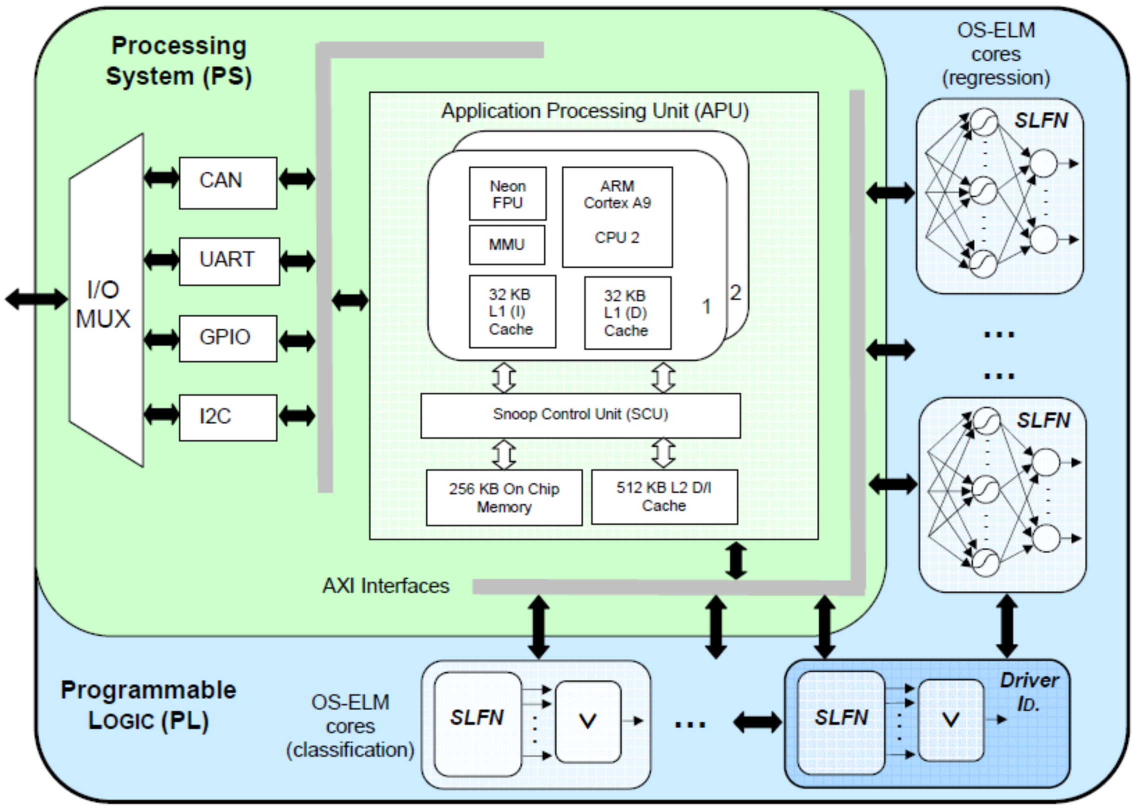


Fig. 3 Block diagram of the HW/SW platform for individualized driver assistance intended for a programmable SoC implementation. Two types of configurable OS-ELM cores are availble: regression and classification cores. The regression cores implement a SLFN; the classification core is an SLFN followed by a maximum module ($\vee$) that activates the identified class. The driver identification module is a particular application of the second type.

programmable SoC of the Zynq device family [36,37]. The internal architecture of this programmable SoC comprises two main parts: a Processing System (PS) that holds the SW partition, and Programmable Logic (PL) where the HW partition is implemented. The PS is built around a dual-core ARM Cortex-A9 hard-processor which is the kernel of the application processor unit (APU). The APU also contains a pair of floating point units (FPU), memory management units (MMU), cache memories, and on-chip memory. The PL part is a piece of FPGA comprising typical reconfigurable resources (i.e. logic blocks, interconnections, and I/O blocks), as well as special resources such as high performance configurable DSP cores and Random Access Memory (RAM) blocks. Similar programmable SoCs are offered by other programmable logic manufacturers [38].

The suitability of HW/SW technologies to implement the OS-ELM driver assistance system is concerned with the distinctive characteristics of the main algorithms that it comprises: the SLFN and the learning algorithm. The former involves very regular and repetitive computations (see Fig. 1), while the latter is an inherently irregular algorithm. It is well known that to obtain efficient HW/SW architectures, the regular and recurrent computations should be implemented in the HW partition, while the irregular or less frequent computations are better suited to the SW partition [39]. In consequence, a well-chosen HW/SW partition consists in implementing the SLFN as hardware on the PL, while the learning algorithm is developed as software on the PS. In addition, the PS supports the whole system monitoring and control, as well as the native applications such as peripheral management. The proposed HW/SW partition provides high-speed for real-time operation and exploits the resources of both the FPGA and the ARM processors. More precisely, the FPGA resources allow the development of high-performance parallel OS-ELM cores for personalized driver assistance, and the high numerical precision of the ARM processors ensures the proper behavior of the corresponding learning algorithms.

The PS/PL communication is performed by means of standard Advanced eXtensible Interfaces (AXI) connections. The system receives information from the measurement units installed in the car, from the user interface, and eventually from the environment (i.e. infrastructures). This information would be continuously monitored, while different kinds of signals, mainly personalized settings and control signals, are generated by the OS-ELM cores attached to the PS. A detailed implementation example is presented in Section 4.

3.1 The Processing System: Software Partition

The SW partition, built on the dual-core ARM processor, is in charge of real-time system monitoring and control including: management of I/O interfaces, management of OS-ELM co-processors, and computation of high-level input features (i.e. arithmetic operations performed on the raw input signals such as: simple scaling functions, statistical functions or spectral components, among others). However, the most computationally demanding task developed in the SW partition is the OS-ELM learning algorithm. Both training modes are available: batch training using previously acquired data according to Eqs. (1)-(6), and online training using chunks of real-time data by means of Eqs. (9)-(10). Whenever a new driver is accepted into the authorized-driver group, the PS trains (or retrains) the driver identification module as well as the whole set of personal applications for that driver. Then, the ARM processor (master) initiates the transference of parameters, weights and biases, to the PL (HW partition) where they are stored in the RAM memory of the corresponding OS-ELM core. The PS is able to meet the performance and precision requirements of the ELM algorithm thanks to the floating point units and the cache memories attached to the ARM processor (see Fig. 3). The portability of the code, developed using C programming language, has been taken into account with the aim of facilitating the migration of the developments to other platforms or operating systems.

3.2 Communication Interfaces

The communication between the PS and the PL is performed by means of AXI interfaces. In particular, the AXI4-Lite bus is used to communicate the PS with the OS-ELM co-processors. This simple memory-mapped protocol is able to transfer an address and a single-data word using the PS/PL bridge (see Fig. 2). The communication between the PS and the external interfaces is achieved by means of the I/O multiplexer (MUX). The mapping between external peripherals and the device pins can be defined as required in order to implement standard communication interfaces and general purpose I/O (GPIO). The platform architecture depicted in Fig. 3 shows a subset of available standard peripherals such as the UART (Universal Asynchronous Receiver Transmitter) for low-rate serial communication, the I2C (Inter-IC) bus accepted by most IC manufacturers, and a typical automotive I/O interface: the CAN (Controller Area Network) bus.

3.3 Programmable Logic: the Hardware Partition

The configurable HW partition comprises a configurable number of parallel ELM co-processors connected with the PS by means of AXI-Lite interfaces. Each co-processor is able to perform the computation of the OS-ELM neural network much faster than a software embedded solution. To fulfil this objective, two main aspects have been exploited. First, the parallel and regular nature of neural networks allows the implementation of highly parallel architectures. Second, the availability of resources for efficient signal processing in current programmable SoC device families, such as the DSP cores, greatly eases the development of high-performance power-efficient applications. An improved version of a preliminary architecture, previously reported in [40], is optimized to fit the computational requirements of actual applications in the automotive sector. The co-processor cores have been conceived as Intellectual Property (IP) modules. They are VHDL modules that can be sized in several dimensions by means of GENERIC parameters (i.e. word-length, number of inputs, number of outputs, and number of neurons in the hidden layer). Recently several authors have proposed special purpose IP-cores for ELM with the aim of extending the range of application of this paradigm [41,42,43].

3.3.1 Hidden neurons and output neurons

The architecture of the SLFN is composed of two kinds of neurons: hidden neurons (i.e. random neurons) and output neurons. Both of them have been efficiently implemented using DSP cores [44]. These hard cores are able to perform sums of products faster than their logic-based counterparts. Fig. 4(a) depicts the block scheme of a hidden neuron (see Eq. (2)), where the random weights are generated out of the chip and further stored in a ROM. The accumulator is initialized with a random bias, and then, a burst of n products (i.e. input data and random weights) are sequentially added. The pipelined sum of products is performed in $(n+1)$ clock cycles. Then, the content of the accumulator is passed through the activation function (a sigmoid function) which is implemented using a ROM block. The output of the hidden neuron is used to address the ROM, where the value of the activation function is read in a single clock cycle. The sigmoid function is costly to implement in digital hardware because it requires the calculation of an exponentiation and a division. To avoid this problem, a number of approximation techniques have been proposed over the years [45]. The most commonly used in FPGA-based hardware implementations is the LUT method. It consists in using LUTs to implement a ROM memory where the values of the sigmoid function are stored. The memory size and word-length should be selected in order to guarantee the required accuracy, but without

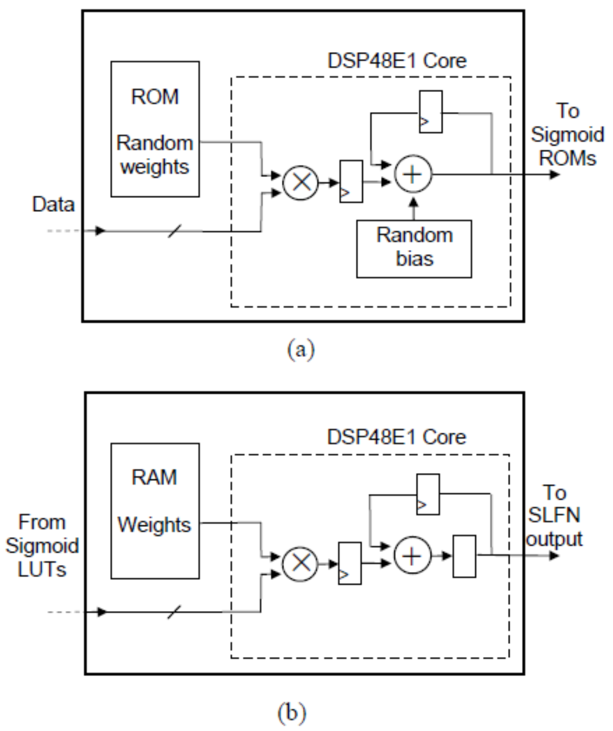


Fig. 4 DSP-based neuron implementation: (a) Neuron of the hidden layer (random neuron), and (b) neuron of the output layer. Both hidden and output neurons are implemented by means of DSP hard cores.

exceeding the availability of LUTs. In this way, fixed-point simulations of the target application can be conducted with the aim of selecting a suitable word-length.

Fig. 4(b) shows the schematic of an output neuron (see Eq. (1)). As can be seen, it is similar to the hidden neuron, but it uses a RAM (instead of a ROM) with the aim of enabling OS-ELM adaptation; distributed RAM memories are used to provide faster access to the data. Moreover, the output neurons are slightly simpler than the hidden neurons because they do not need a bias register and the sum of products is directly driven to the output, without passing through an activation function.

3.3.2 Layered structure of the SLFN

The hardware implementation of the OS-ELM co-processor core is a direct synthesis of the layered architecture depicted in Fig. 1. The suitability of the SLFN for parallel neuron computation in each layer has been highly exploited with the aim of reducing the core latency. The input layer transfers the n inputs to the neurons in the hidden layer, one input per clock cycle. Meanwhile the L hidden neurons are concurrently computed in $n+2$ cycles, where the sigmoid activation cycle is included. Afterwards the output neurons are computed in parallel lasting $L+1$ cycles, independently of the number of network outputs. In sum, the computation of an SLFN with an $n-L-m$ topology requires only $n+L+3$ clock cycles. A sequential software solution, using a one cycle per instruction processor, like the ARM hard-core, would require at least $2nL+2mL+L$ clock cycles to compute only the linear part of the SLFN (i.e. sums and products). In practice, the number of cycles is considerably greater, mainly because of the calculus of the Sigmoid function and the resource management involved in nested computations. It is clear that the acceleration of the SLFN computation using the above hardware approach is outstanding. Even more, since all instances of the OS-ELM core are able to process data concurrently, the achieved acceleration increases proportionately to the number of cores, to the extent that input features are available.

3.4 Implementation of the configurable SoC architecture

The proposed architecture has been implemented using the ZC702 development board which features a XC7Z020 CLG484-1 SoC of the Xilinx Zynq family. The device embeds a dual-core ARM Cortex-A9 operating up to 866 MHz. It provides 53,200 look-up tables (LUTs); 106,400 flip-flops; 220 DSP slices (i.e. 18x25 multiply-accumulate MAC- unit); and 140 RAM memory blocks of 36 Kbits each. It is worth noting that this is a low-end device, the third of seven available sizes, while the largest device of the family is 10 times larger than this one and the performance of its ARM dual-core goes up to 1GHz.

Several parallel ELM co-processors, with or without online sequential learning capability, can be implemented in the hardware partition of the programmable SoC (PL) for the deployment of a wide variety of in-vehicle applications, including classification problems, complex function modelling, and low-level nonlinear control. The driver identification module is a particular instantiation of a classification-type OS-ELM core.

Table 1 summarizes the resources required to synthesize different SLFN ELM topologies and the corresponding maximum clock frequencies. These results were obtained using a 10-bit twos complement fixed-point fractional data format. With the aim of providing straightforward comparative information, results corresponding to single-output topologies are shown. The SLFN co-processors with 16 inputs and 64 hidden neurons use approximately 30% of the resources available in the XC7Z020 programmable SoC, while a larger co-processor based on the next power-of-two topology (i.e. 32-128-1 SLFN ELM) would require slightly more than 100% of available LUTs, 10% of flip-flops, and 60% DSPs. The latter topology, or even more complex ones, could be implemented using larger SoCs of the same family. Concerning timing considerations, all of the networks in Table I can be computed in less than 0.5 microseconds.

Table 1 Resource Usage and Maximum Operation Frequency of the OS-ELM Co-Processor For Regression Using the XC7Z020 CLG484-1 SOC

ELM topology[a]	LUTs	Flip-flops	DSPs	Maximum frequency (MHz)
2-8-1	2,403 (4%)	577 (<1%)	9 (4%)	186
4-16-1	5,128 (9%)	1,126 (1%)	17 (7%)	185
8-32-1	9,801 (18%)	2,320 (2%)	33 (15%)	183
16-64-1	18,539 (34%)	4,527 (4%)	65 (29%)	179

[a]With the aim of providing straightforward comparative information, results corresponding to single-output topologies are shown.

The resource usage summary provided in Table 1 highlights the relevance of DSP slices for the development of the proposed approach. The availability of DSP resources is what really limits the number and complexity of the parallel OS-ELM cores that can be implemented in the PL. Given that both the hidden and the output neurons use a single DSP slice (see Fig. 4), the implementation of each OS-ELM core lasts $L+m$ DSP slices, with L being the number of hidden nodes and m being the number of outputs. Let us now consider the largest device of the Zynq family, the XCZ100, that features as many as 2020 DPSs. The whole set of DSPs can either be assigned to the implementation of a single OS-ELM core or they can be distributed with the aim of implementing several heterogeneous instances of the core. Although the scalability of the core is limited by the availability of DSP resources, larger SLFN are not usually required to implement driver identification and driver assistance personalization. However, if a more complex architecture were necessary, an MPSoC (multiprocessing SoC) such as the Zynq UltraScale+ could be used [46]. These sophisticated devices increase the PL resources and extend the processing capabilities of the PS to deal with cutting-edge applications.

As can be seen in Fig. 4, the OS-ELM core embeds different kinds of memories: sigmoid ROM, random weights ROM and output weights RAM. All of these are implemented using LUTs with the aim of favoring the circuit speed (i.e. using distributed memory, instead of block RAM, reduces propagation delays). Table 2 summarizes the LUT resources used to implement each memory type. Both ROM and RAM memories could also be implemented using Block RAM. However, addressing this kind of memories introduces additional propagation delays, thus reducing the maximum operation frequency.

Both the HW partition and the SW partition are fully scalable, that is to say, only minor changes have to be made to adapt the design to a variety of applications for regression and/or classification problems. The configurable co-processor core has been developed with the aid of the Xilinx ISE Design Suite. The software partition and the whole system integration have been made by means of the Xilinx SDK and Vivado Design Suite [47].

Table 2 LUT Usage: Implementation of ROM and RAM Modules Using the XC7Z020 CLG484-1 SOC

ELM topology	Total LUTs	Sigmoid function ROM	Random weights ROM	Output weights RAM[a]	LUTs used as logic
2-8-1	2,403	1,416	8	11	968
4-16-1	5,128	2,832	32	11	2,253
8-32-1	9,801	5,664	160	11	3,966
16-64-1	18,539	11,328	384	11	6,816

[a]The Zynq device family features 6-input LUTs. In consequence, up to 64 memory words can be stored using a single LUT per bit (i.e. the RAM word-length is 11 bits).

4 Application

In this section, a particular set of driver assistance features, suitable for current levels of driving automation, is presented. The multipurpose personalized applications were developed using the architecture and resources depicted in Fig. 3. A driver identification strategy based on driving behavior signals has been selected because of the non-intrusive nature of these signals and the low cost of additional equipment. As will be seen, the driver identification module is able to model individual differences in the driving style of a group of drivers and identify the driver in real-time. The versatility of the developed platform and the plasticity of the OS-ELM cores to model different kinds of in-vehicle applications are evidenced by this case application. Three kinds of personalized driver assistance features are implemented: a security feature for anti-theft impostor detection, a safety application for ACC driving assistance, and an air recirculation application as a comfort application. First, the Uyanik data collection, used to develop the proposed application, is introduced.

4.1 Data collection

The data set was collected in Istanbul using the Uyanik instrumented car. It is a sedan car equipped with different sensors and measurement units [7,48]. The car route is around 25 km (about 40 minutes), and includes different kinds of roads and traffic sections: city, very busy city, highway, highway with less traffic, and a university campus. A subset of the recording sessions consisting of 11 drivers was chosen; recordings with missing values or incomplete information were discarded. The complete data set includes audio and video recordings, CAN-bus signals, pedal-sensor recordings, a frontal laser scanner, and an inertial measurement unit (IMU). However, in this application neither video nor audio signals were considered because of their intrusive nature: some drivers consider that these types of signals invade their privacy. All signals are handled jointly, which requires a resampling of the datastreams to the highest frequency of 32 Hz to homogenize the time series of available signals (see Table 3).

4.2 Driver identification module

The driver identification module (see Fig. 3), or more precisely its implementation using OS-ELM, is the key element that endows the HW/SW platform with the ability to provide personalized assistance to the driver. The most relevant design steps of this module are presented, and obtained experimental results are shown.

4.2.1 Design and multi-objective optimization of the system

A set of high level features, which have been successfully used for modeling driving behavior, has been considered as potential input features [7]. Table 3 summarizes the set of low level variables (i.e. time series), and 42 high level features: mean value (time domain), energy (frequency domain), and Cepstral coefficients (Quefrency domain). The features are computed over 128-second windows (i.e. 4096 samples) with 1-second shift (i.e. 32 sample shift). That is to say, with an overlapping of 127 seconds between consecutive windows. Reducing the number of features (i.e. the dimensionality of the problem) is important in machine learning, mainly when the machine core is to be embedded in a single chip. Both computation time and storage resources can be drastically reduced with an adequate selection of the subset of useful signals and variables. The multi-objective optimization algorithm introduced in Section 2 was used with the aim of achieving an optimal feature selection (see Fig. 2, first step: topology design).

Figure 5 shows a Pareto front obtained using the 3-dimensional vector of functions in Eq. (11). It can be seen that the points of the front are always close to the minimum value of at least one of the 3 objectives (i.e. number of inputs, number of hidden neurons, and testing error) in an attempt to minimize the objective functions. The front maintains a high diversity of solutions, containing points in a wide range of values. For example, point A=(3, 2, 79.6%), in one end of the front, provides a very low number of inputs and a low number of hidden neurons. However, the error rate is as high as 79.6%. At the other end of the front, point B=(11, 63, 7.8%) has a low error rate, 7.8%, but the ELM neural network needs as much as 11 input features and 63 hidden neurons.

Finally, point C=(8, 31, 11.7%) is a suitable trade-off solution: a simple topology with 8 inputs and 31 hidden neurons is able to provide 88.3% of recognition performance. Bearing in mind that a digital hardware approach is to be developed, a convenient decision would be to select power-of-two topologies that favor resource efficient implementations. Therefore, an SLFN with 8 inputs (boldface numbers in Table 3) and 32 hidden neurons, instead of 31, has been selected to implement the driver identification OS-ELM. As can be seen, gas pedal pressure (features 7, 21 and 35) and brake pedal pressure (features 20 and 34) are the most informative driving behavior signals for developing the identification module.

4.2.2 Experimental results: real-time driver identification

The optimized driver identification system has been extensively tested using the selected features and the Uyanik data set. The whole set of 11 drivers, as well as within subgroups of *m*=2, 3, 4, and 5 drivers, has been evaluated with the aim of recreating real-life situations. In this evaluation, two thirds of the data were intended to train the system, and the remaining one third was saved for testing. The average driver identification rates take into account all possible subgroups of drivers, in each category, using the subset of 8 selected features (see Table 3).

The mean identification rate for the 11-driver set is 84.4%; within groups of 5 drivers the system achieves identification rates of 91.3%; within groups of 4 drivers the rate is 93.0%; the prediction for groups of 3 drivers is 94.9%; and with two drivers the rate rises to 96.9%

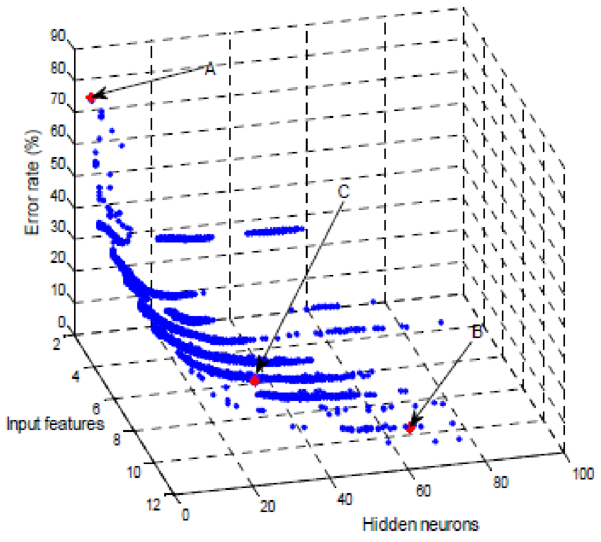


Fig. 5 Pareto front. Points A and B have minimun values in the 2nd and 3rd objective, respectively, while point C provides a trade-off solution.

Table 3 Driver Identification: Feature Selection. High Level Features are Derived from Analysis Windows (128 Sec)

Features	Signals (time series)	Time	Frequency	Quefrency
CAN-bus	SWA: Steering wheel angle	1	15	29
	SWS: Steering wheel speed	2	16	**30**
	VS: Vehicle speed	3	17	31
	PGP: Percent gas pedal	4	18	32
	ERPM: Engine RPM	5	19	33
Pressure sensors	BP: Brake pedal pressure	6	**20**	**34**
	GP: Gas pedal pressure	**7**	**21**	**35**
IMU unit	RR: Roll rate	**8**	22	36
	PR: Pitch rate	**9**	23	37
	YR: Yaw rate	10	24	38
	XACC: X axis accelerometer	11	25	39
	YACC: Y axis accelerometer	12	26	40
	ZACC: Z axis accelerometer	13	27	41
Laser	d_90: Distance to obstacle	14	28	42

The boldface numbers correspond to the 8 features selected using a multi-objective optimization algorithm, to develop both the driver identification module and the anti-theft application.

(see red bars in Fig. 6). These results outperform previous experiments performed with the same data set using a reduced number of features [7,8,9]. It is worth mentioning that recently superior identification rates have been achieved using AdaBoost [10] and gaussian mixture models [11]. However, the former involves a pre-training step on a server, while the latter significantly increases the number of selected features (i.e. 40 features instead of 8 features).

Figure 7 depicts a series of experiments performed with the aim of assessing real-time driver identification. Prediction is assessed every minute throughout the last 10-minute section of each driver recording, which is in the testing part of the route for all drivers (i.e. unseen data). Each subplot represents the accumulated identification estimations over time (in minutes). As can be seen, the system succeeds in identifying the drivers right from the first decision window (3 minutes), except for driver d03. Even for this driver, the system is able to achieve the correct prediction after a few windows along the first minutes of the route. Moreover, for drivers d01, d09, d10, and d11, prediction is about 100% at almost any point of the evaluated sections.

4.2.3 Module implementation and timing performance

The driver identification system has been implemented using the XC7Z020 Zynq device. Table 4 summarizes the hardware resources required to implement the (8-32-11) optimized topology using a 10-bit twos complement fixed-point fractional data format. As can be seen, it uses less than 20% of the resources available in the PL part of this small programmable SoC. Concerning the timing performance, the maximum operation frequency of this core is 183 MHz.

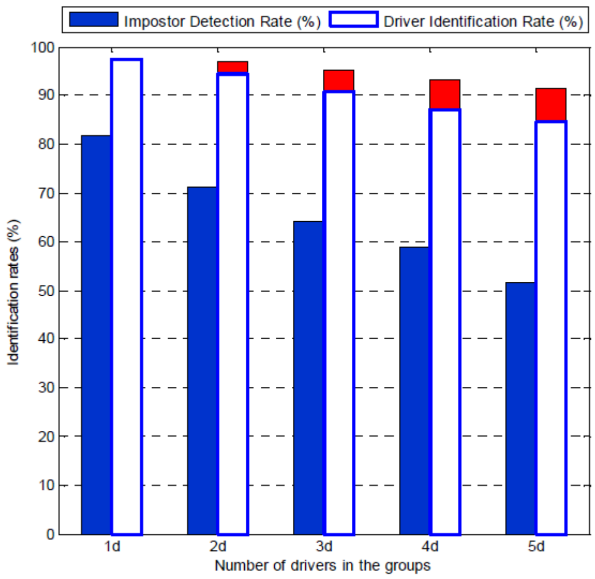


Fig. 6 Red bars in the background represent the average driver identification rates using closed-set models. Blue bars and white bars with blue border lines represent the average impostor detection rate and the driver identification rates using open-set models, respectively.

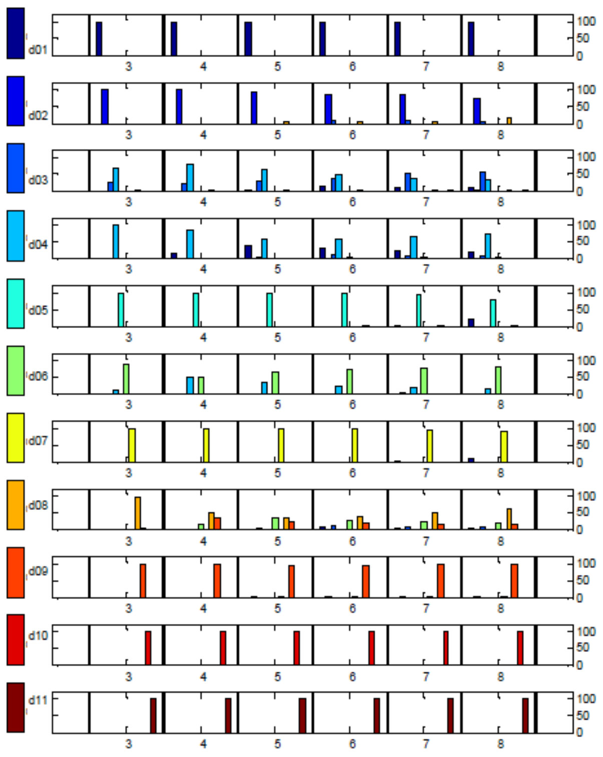


Fig. 7 Identification percentage and estimation over time, in minutes, of OS-ELM system for the last 10 min section of every driver (d01, d02,, d11). A different color is assigned to each driver.

Let us consider both the real-time OS-ELM training stage (i.e. on-chip self-learning) and the real-time driver identification stage (see Fig. 2). The following timing performance figures were obtained with the ARM in the PS operating at 860 MHz, and the OS-ELM co-processor in the PL operating at 100 MHz. The ARM requires 3.2 ms to compute the eight selected features, where feature normalization is included. During the training stage, the system reads input signals and computes the corresponding high level features until all the samples of a chunk are acquired. A new window is read every second (time shift required to get 32 samples with a 32 Hz sampling rate), so a new chunk of K samples is available every K seconds. The system has been successfully tested using different sizes of chunks ranging from $K = 1$ to $K = 400$. In the most time-consuming situation, $K = 400$, the PS is able to perform a learning step (see Eq. (10)) in less than 15.7 ms. After all the chunks have been processed, the trained parameters are sent to the HW partition where they are stored in the internal RAM of the driver identification core (see Fig. 3). The network parameters can be updated whenever a new driver enters the group or upon request of an allowed driver.

After training, the real-time driver identification system can be activated. In this stage, the SW partition performs I/O processing of signals and feature computation, while the HW partition processes the trained SLFN. The core is able

Table 4 Resource Usage of the OS-ELM Co-Processor For Each of the DAS Applications Using the XC7Z020 CLG484-1 SOC

DAS application	ELM topology	LUTs	LUTs memory	Flip-flops	DSPs
Driver Id.	8-32-11	10,623 (19%)	5,945 (11%)	2,894 (2%)	43 (19%)
Security	8-32-12	≈	≈	≈	44 (20%)
Safety	2-8-1	2,403 (4%)	1,435 (2%)	577 (<1%)	9 (4%)
Comfort	5-16-1	5,384 (10%)	2,923 (5%)	1,182 (1%)	17 (7%)

to perform the computation of the SLFN in only 2.2 μs using a 100 MHz clock. The data transfer from the PS to the OS-ELM core and vice versa, using the AXI-lite interface, as well as the computation of the maximum identification rate, is also included in the above timing performance.

4.3 Security application: Anti-theft impostor Detection

Currently a number of cars are equipped with anti-theft utilities such as RFID (radio-frequency identification) devices or personalized cards. However, most of them can be easily hacked. To overcome this problem, the above module can be slightly modified, using very few additional resources, with the aim of increasing vehicle security against thefts.

To accomplish this objective, the module was enhanced with the ability to detect a driver outside a group of authorized drivers: a thief or driver impostor. In addition, high identification performance within the group of known drivers is maintained. Concerning the topology of the driver identification SLFN, a single neuron is to be added to the output layer to account for the rejection class (i.e. impostor). Therefore, the new network has an 8-32-(m+1) topology, where the $(m+1)th$ output represents any driver other than those of the authorized driver set. It is worth mentioning that the previous closed-set driver identification model has been turned into an open-set classification system.

The enhanced model has been comprehensively tested using the previously selected driving behavior signals for subgroups of m=1, 2, 3, 4, and 5 drivers. Each experiment considers a group of m genuine drivers. Another driver for each test case is marked as the impostor and excluded from training. The network is trained with data of the m genuine drivers of the group, and data from the rest of the drivers in the collection, which are labeled as others. For testing, the system computes the prediction using unseen data of the genuine drivers and of the driver labeled as impostor, who is unknown for the classifier. The identification rates (see Fig. 6) remain above 90% for groups of two and three drivers, and around 85% for groups of four and five drivers. On the other hand, the impostor detection rate is above 80% when the car has a single allowed driver, but this rate decreases inversely to the number of allowed drivers in the group. Although the impostor detection rate is above 50% in all the studied cases, the reliability of the system could be improved using a fusion of driving behavior signals and non-hackable information (e.g. voice signals). Timing performance and resource usage are similar to those obtained in the previous closed-set driver identification implementation (see Table 4).

4.4 Safety Application: Longitudinal Driving Assistance

Several longitudinal driving assistance (LDA) systems are currently available in commercial vehicles for assisting the driver in car-following scenarios, namely adaptive cruise control (ACC), and forward collision warning/avoidance (FCW/FCA) [1]. The aim of these DAS is to maintain a safe and comfortable distance between vehicles in car-following scenarios reducing, thus, driver workload and traffic accidents. Although the performance of these systems has been steadily improved in recent years, personalization of LDA would enhance the driving experience even more.

Some studies reveal that temporal distance variables such as time headway (THW), time-to-collision (TTC), and the inverse of TTC (i.e. TTCi) are useful variables for modeling car-following behavior, [49]:

$$THW = \frac{d}{v}, \tag{12}$$

$$TTC = \frac{d}{v_r}, \qquad TTCi = TTC^{-1}, \tag{13}$$

with d being the distance between the host vehicle and the leading vehicle, and with v and v_r being the speed of the host vehicle and its relative speed to the leading vehicle, respectively.

An LDA system, which can be adapted to driver behavior, is proposed in [14]. The authors developed a personalized driver model, based on the desired temporal distance variables THW and TTCi, providing driving performance closer to the driver characteristics than typical ACC control. However, the above model assumes constant parameters. A more natural driver behavior model is presented in [50] where the relationship between THW/TTCi and the desired car accelerations is modeled using a two-layer BP-ANN. However, self-learning using BP learning techniques is unsuitable for in-car integration. On the contrary, ELM-based learning is fast and easy to implement, even for a time-critical application, as is a personalized ACC DAS.

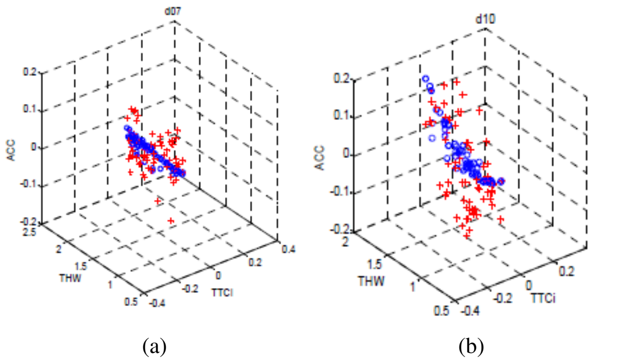


Fig. 8 Acceleration as a function of TTCi and THW for two different drivers: real data: + and OS-ELM model: o. The data sequences of car-following scenarios have been extracted from the Uyanik data set. See (a) driver labeled d07, and (b) driver d10 in Fig. 7

The OS-ELM core has been used to develop a personalized acceleration model, suitable for the implementation of a self-learning ACC control. Temporal distances Eqs. (12) and (13) are obtained from the vehicle speed (VS) and the distance to obstacle (d_90), while the desired acceleration is provided by the accelerometer of the IMU unit (see Table 3). The data sequences of car-following scenarios have been extracted from the time series of the Uyanik data set. In particular, the ACC function is defined as the host car following a fixed leading vehicle for more than 15 seconds without changing lanes [49].

A sweep in the number of hidden nodes, L, has been performed with the aim of showing the relationship between this parameter and the testing error. A trade-off solution complexity/performance is achieved with a straightforward (2-8-1) topology. The OS-ELM acceleration model has been trained using two thirds of the car-following data while the remaining data have been used to test the network. In view of the reduced amount of available data corresponding to car-following situations, batch ELM (Eqs. (1)-(6)) has been applied to train the system. Figure 8 provides experimental results corresponding to two different drivers in the Uyanik data set. In particular, the acceleration model of the drivers labeled d07 and d10 (see Fig. 7) is provided. The root mean squared error (RMSE) is 0.07 for driver d07 and 0.12 for driver d10. As can be seen, both drivers present mean TTCi close to zero in car-following scenarios, while mean THW is larger for d07 (1.59 s) than d10 (1.33 s) indicating that d10 has a more aggressive driving style. Moreover, the OS-ELM model has the effect of filtering the data, providing a smoothed relationship between acceleration and THW, while retaining the personal style of the drivers.

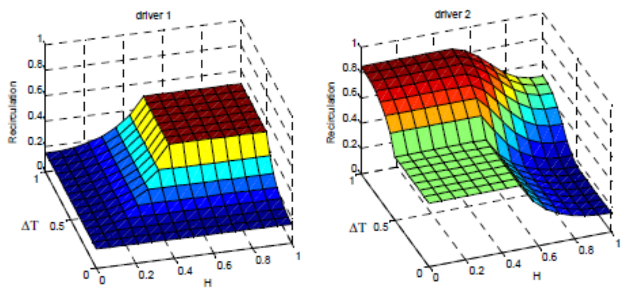
The implementation of the core requires a reduced number of resources (see Table 4) and provides up to 186 MHz operation frequencies. Assuming the conservative clock frequency for the hardware partition of 100MHz, the core is able to provide the desired car acceleration in less than 0.8 μs. This performance is compatible even with the most cutting edge Lidar technology that is able to provide a measurement of the distance to the leading vehicle every 10 μs.

4.5 Comfort Application

Finally, concerning in-cabin features, the embedded system should be able to recognize the driver and anticipate his/her preferences and needs in a personalized and adaptable way. Although comfort features are neither safety critical nor time critical applications, the major challenge of these kinds of applications deals with the complexity of human behavior modeling. A personalized model for each configurable feature in the car can be obtained by monitoring the actions of each authorized driver and learning through observation. In particular, with the aim of illustrating the modeling capability of OS-ELM, an air recirculation feature has been chosen.

Many recent car models incorporate automatic systems with sensors that monitor in-cabin moisture and oxygen levels, switching between the recirculating and fresh air modes depending on contaminant particles or energy savings and efficiency [51,52]. However, the driver would feel more comfortable with a personalized air re-circulation model that keeps in mind his/her priorities. Five input variables have been considered: humidity (H), air quality inside the car (Q), difference between external and internal temperature (ΔT), air flow direction (D), and car velocity (VS). Depending on these variables, each driver will act in a different way with the air recirculation feature.

Since no actual comfort data are available in the Uyanik data collection, a set of linguistic fuzzy rules has been developed for each driver. Fig. 9 shows the generated surfaces of the air recirculation aperture decision for two different drivers, depending on H and ΔT variables, with Q, D, and VS being exactly in the middle of their respective ranges. As can be seen, driver 1 prioritizes fresh air over the rest of the variables, whereas for driver 2, the recirculation mode is preferred when the difference of temperature is high but, it is not desired if the humidity is high. A set of data that simulates real situations has been extracted with the aim of training an OS-ELM core. The modeling capability of OS-ELM cores to approximate the surfaces depicted in Fig. 9 has been analyzed and a topology with 16 hidden neurons has been chosen. It provides around 10% and 6% accuracy for driver 1 and 2, respectively. As can be seen in Table 4, where an implementation resource summary is shown, this solution provides a well-balanced trade-off between resources and accuracy.



(a) (b)

Fig. 9 Graphical representation of the desired air re-circulation aperture for two different drivers. Re-circulation equal to 0 (totally closed) means only fresh air and 1 (totally open) means only cabin air is re-circulated.

5 Conclusion

In this work, a holistic approach to the challenge of car personalization is presented. The kernel of the proposal is a versatile hardware/software platform for personalized driver assistance using a programmable system-on-chip (SoC). The system, based on online sequential extreme learning machines (OS-ELM), is able to recognize the driver and personalize the behavior of the car. The most remarkable feature of the OS-ELM platform is its ability to train personalized assistance functionalities in real-time, while driving normally.

The proposed approach exploits the synergy between car cybernetics and machine learning with the aim of deploying a wide variety of in-vehicle personalized features including safety, security and comfort applications. Although this perspective is conceived with the aim of improving the quality of individual mobility, it can easily be extended to account for the requirements of particular groups of drivers, for example, those with high accident rates or the particularly vulnerable (e.g. young or elderly drivers). In addition, different aspects of driving such as, for example, improving fuel-efficient driving, can also be dealt with.

In sum, due to the plasticity and scalability of the OS-ELM algorithm and the programmable nature of SoCs, the proposed platform is flexible enough to cope with the incremental changes that the new generation of vehicles is demanding. The platform architecture is able to evolve and to adapt to the requirements of present and forthcoming levels of driving automation, thus reducing costs and time to market, while enhancing vehicle safety and reliability.

In future work, the capabilities of the platform will be enhanced with configurable Deep-ELM cores with the aim of providing support to complex cutting-edge applications demanding more flexibility and accuracy to describe complicated human actions, including real-time audio or video processing.

Acknowledgements This work was supported in part by the Spanish Ministry of Economy and Competitiveness (MINECO) under Grant TEC2013-42286-R, and by the Basque Country University UPV/EHU under Grant PPG17/20.

Conflict of Interest: The authors declare that they have no conflict of interest.

References

1. Bengler, K., Dietmayer, K., Farber, B., Maurer, M., Stiller, C., Winner, H.: Three decades of driver assistance systems: Review and future perspectives. IEEE Intelligent Transportation Systems Magazine **6**(4), 6–22 (2014). DOI 10.1109/MITS.2014.2336271
2. Elbanhawi, M., Simic, M., Jazar, R.: In the passenger seat: Investigating ride comfort measures in autonomous cars. IEEE Intelligent Transportation Systems Magazine **7**(3), 4–17 (2015). DOI 10.1109/MITS.2015.2405571
3. Ford Motor Company: Ford and Intel Research Demonstrates the Future of In-Car Personalization and Mobile Interior Imaging Technology (2014). URL `https://media.ford.com/content/fordmedia/fna/us/en/news/2014/06/25/ford-and-intel-research-demonstrates-the-future-of-in-car-person.html`
4. Wu, J.D., Ye, S.H.: Driver identification based on voice signal using continuous wavelet transform and artificial neural network techniques. Expert Systems with Applications **36**(2), 1061–1069 (2009). DOI http://dx.doi.org/10.1016/j.eswa.2007.11.003. URL `http://www.sciencedirect.com/science/article/pii/S0957417407005271`
5. Riener, A., Fersha, A.: Supporting implicit human-to-vehicle interaction: Driver identification from sitting postures. In: Proc. of the ISVCS, pp. 22–24. Dublin, Ireland (2008)
6. Qian, H., Ou, Y., Wu, X., Meng, X., Xu, Y.: Support vector machine for behavior-based driver identification system. Journal of Robotics **2010, 11 pages** (2010). DOI doi:10.1155/2010/397865. URL `https://www.hindawi.com/journals/jr/2010/397865/`
7. Öztürk, E., Erzin, E.: Driver status identification from driving behavior signals. In: J. Hansen, P. Boyraz, K. Takeda, H. Abut (eds.) Digital Signal Processing for In-vehicle Systems and Safety, pp. 31–55. Springer Business-Science (2012)
8. Martínez, M.V., del Campo, I., Echanobe, J., Basterretxea, K.: Driving behavior signals and machine learning: A personalized driver assistance system. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pp. 2933–2940 (2015). DOI 10.1109/ITSC.2015.470
9. Martínez, M.V., Echanobe, J., del Campo, I.: Driver identification and impostor detection based on driving behavior signals. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pp. 372–378 (2016). DOI 10.1109/ITSC.2016.7795582
10. Jafarnejad, S., Castignani, G., Engel, T.: Towards a real-time driver identification mechanism based on driving sensing data. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC) (2017). DOI 10.1109/ITSC.2017.8317716
11. Jafarnejad, S., Castignani, G., Engel, T.: Revisiting gaussian mixture models for driver identification. In: 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES) (2018). DOI 10.1109/ICVES.2018.8519588
12. Moreira-Matias, L., Farah, H.: On developing a driver identification methodology using in-vehicle data recorders. IEEE Transactions on Intelligent Transportation Systems **18**(9), 2387–2396 (2017). DOI 10.1109/TITS.2016.2639361
13. Butakov, V., Ioannou, P.: Driving autopilot with personalization feature for improved safety and comfort. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pp. 387–393 (2015). DOI 10.1109/ITSC.2015.72
14. Wang, J., Zhang, L., Zhang, D., Li, K.: An adaptive longitudinal driving assistance system based on driver characteristics. IEEE Transactions on Intelligent Transportation Systems **14**(1), 1–12 (2013). DOI 10.1109/TITS.2012.2205143
15. Butakov, V.A., Ioannou, P.: Personalized driver/vehicle lane change models for adas. IEEE Transactions on Vehicular Technology **64**(10), 4422–4431 (2015). DOI 10.1109/TVT.2014.2369522
16. Lexus: Programming Memory Seats and Pairing Smart Key in Lexus (2016). URL `http://www.whylexus.com/programming-memory-seats-and-pairing-smart-key-in-lexus`
17. Buick: Vehicle Personalization (2017). URL `http://www.buiclub.com/info-1733.html`
18. Chen, Y., Li, L. (eds.): Advances in Intelligent Vehicles: Intelligent Systems Series. Academic Press (2014)
19. Ohn-Bar, E., Trivedi, M.M.: Looking at humans in the age of self-driving and highly automated vehicles. IEEE Transactions on Intelligent Vehicles **1**(1), 90–104 (2016). DOI 10.1109/TIV.2016.2571067
20. Huang, G.B., Zhou, H., Ding, X., Zhang, R.: Extreme learning machine for regression and multiclass classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **42**(2), 513–529 (2012). DOI 10.1109/TSMCB.2011.2168604
21. Ding, S., Xu, X., Nie, R.: Extreme learning machine and its applications. Neural Computing and Applications **25**(3), 549–556 (2014). DOI 10.1007/s00521-013-1522-8. URL `https://doi.org/10.1007/s00521-013-1522-8`
22. Tang, J., Deng, C., Huang, G.B.: Extreme learning machine for multilayer perceptron. IEEE Transactions on Neural Networks and Learning Systems **27**(4), 809–821 (2016). DOI 10.1109/TNNLS.2015.2424995
23. y. Liang, N., b. Huang, G., Saratchandran, P., Sundararajan, N.: A fast and accurate online sequential learning algorithm for feedforward networks. IEEE Transactions on Neural Networks **17**(6), 1411–1423 (2006). DOI 10.1109/TNN.2006.880583
24. Renesas Electronics Coorporation: Advanced Driver Assistance System (ADAS) (2017). URL `https://www.renesas.com/en-us/solutions/automotive/adas.html`
25. NVIDIA Coorporation: NVIDIA DRIVE PX Scalable Supercomputer for Autonomous Driving (2017). URL `http://www.nvidia.com/object/drive-px.html`
26. NXP Semiconductors: BlueBox: Autonomous Driving Platform (S32VLS2-RDB) (2017). URL `http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/s32-arm-processors-microcontrollers/bluebox-autonomous-driving-platform-s32vls2-rdb:S32VLS2-RDB`
27. Mobileye: The Evolution of EyeQ (2017). URL `http://www.mobileye.com/our-technology/evolution-eyeq-chip/`
28. Texas Instruments: TDAx ADAS SoCs (2017). URL `https://www.ti.com/processors/automotive-processors/tdax-adas-socs/overview.html`
29. Gage, T., Morris, J.: The coming revolution in vehicle technology and its big implications. Xcell Journal **92**, 38–45 (2015). URL `https://www.xilinx.com/publications/archives/xcell/Xcell92.pdf`
30. Johnson, A.P., Liu, J.X., Millard, A.G., Karim, S., Tyrrell, A.M., Harkin, J., Timmis, J., McDaid, L.J., Halliday, D.M.: Homeostatic Fault Tolerance in Spiking Neural Networks: A Dynamic Hardware Perspective. IEEE Transactions on Circuits

and Systems-I: Regular Papers **65**(2), 687–699 (2018). DOI 10.1109/TCSI.2017.2726763

31. Liu, J.X., Harkin, J., Maguire, L.P., McDaid, L.J., Wade, J.J.: SPANNER: A Self-Repairing Spiking Neural Network Hardware Architecture. IEEE Transactions on Neural Networks and Learning Systems **29**(4), 1287–1300 (2018). DOI 10.1109/TNNLS.2017.2673021
32. Feng, G., Huang, G.B., Lin, Q., Gay, R.: Error minimized extreme learning machine with growth of hidden nodes and incremental learning. IEEE Transactions on Neural Networks **20**(8), 1352–1357 (2009). DOI 10.1109/TNN.2009.2024147
33. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes. The Art of Scientific Computing, third edn. Cambridge University Press (2007)
34. Pryke, A., Mostaghim, S., Nazemi, A.: Heatmap Visualization of Population Based Multi Objective Algorithms, pp. 361–375. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). URL `https://link.springer.com/chapter/10.1007/978-3-540-70928-2_29`
35. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002). DOI 10.1109/4235.996017
36. Xilinx Inc.: All Programmable SoC with Hardware and Software Programmability (2017). URL `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`
37. Crockett, L.H., Elliot, R.A., Enderwitz, M.A., Stewart, R.W.: The Zynq Book. University of Strathclyde (2015). URL `http://www.zynqbook.com/`
38. Intel Corporation: Intel User-Customizable SoC-FPGAs (2017). URL `https://www.altera.com/en_US/pdfs/literature/br/br-soc-fpga.pdf`
39. Xilinx Inc.: SDSoC Programmers Guide (2018). URL `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1278-sdsoc-programmers-guide.pdf`
40. Finker, R., del Campo, I., Echanobe, J., Martnez, V.: An intelligent embedded system for real-time adaptive extreme learning machine. In: 2014 IEEE Symposium on Intelligent Embedded Systems (IES), pp. 61–69 (2014). DOI 10.1109/INTELES.2014.7008987
41. Frances-Villora, J., Rosado-Muoz, A., Martnez-Villena, J.M., Bataller-Mompean, M., Guerrero, J.F., Wegrzyn, M.: Hardware implementation of real-time extreme learning machine in fpga: Analysis of precision, resource occupation and performance. Computers & Electrical Engineering **51**, 139–156 (2016). DOI http://dx.doi.org/10.1016/j.compeleceng.2016.02.007. URL `http://www.sciencedirect.com/science/article/pii/S0045790616300222`
42. Bataller-Mompen, M., Martnez-Villena, J.M., Rosado-Muoz, A., Frances-Villora, J.V., Guerrero-Martnez, J.F., Wegrzyn, M., Adamski, M.: Support tool for the combined software/hardware design of on-chip elm training for slff neural networks. IEEE Transactions on Industrial Informatics **12**(3), 1114–1123 (2016). DOI 10.1109/TII.2016.2554521
43. Yeam, T.C., Ismail, N., Mashiko, K., Matsuzaki, T.: Fpga implementation of extreme learning machine system for classification. In: TENCON 2017 - 2017 IEEE Region 10 Conference, pp. 1868–1873 (2017). DOI 10.1109/TENCON.2017.8228163
44. Xilinx Inc.: 7 Series DSP48E1 Slice. User Guide, ug479 (v1.9) edn. (2016). URL `https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf`
45. Bosque, G., del Campo, I., Echanobe, J.: Fuzzy systems, neural networks and neuro-fuzzy systems: A vision on their hardware implementation and platforms over two decades. Engineering Applications of Artificial Intelligence **32**, 283 – 331 (2014). DOI https://doi.org/10.1016/j.engappai.2014.02.008. URL `http://www.sciencedirect.com/science/article/pii/S0952197614000384`
46. Xilinx Inc.: Zynq UltraScale+ MPSoC. Product Tables and Product Selection Guide (2018). URL `https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf`
47. Xilinx Inc.: Hardware Zone (2017). URL `https://www.xilinx.com/products/design-tools/hardware-zone.html`
48. Abut, H., Erdogan, H., Ercil, A., Çürüklü, B., Koman, H.C., Tas, F., Argunsah, A.Ö., Cosar, S., Akan, B., Karabalkan, H., Cökelek, E., Ficici, R., Sezer, V., Danis, S., Karaca, M., Abbak, M., Uzunba, M.G., Eritmen, K., Imamolu, M., Kalaycoglu, C.: Real-World Data Collection with UYANIK, chap. 3, pp. 23–44. Springer US (2009). URL `http://www.es.mdh.se/publications/2852-Real_World_Data_Collection_with_UYANIK`
49. Qi, G., Du, Y., Wu, J., Hounsell, N., Jia, Y.: What is the appropriate temporal distance range for driving style analysis? IEEE Transactions on Intelligent Transportation Systems **17**(5), 1393–1403 (2016). DOI 10.1109/TITS.2015.2502985
50. Wang, J., Li, K., Lu, X.Y.: Comparative Analysis and Modeling of Driver Behavior Characteristics, chap. 6, pp. 159–198. Academic Press (2014)
51. Ostermeier, R., Rühl, G.: Method for controlling a ventilation/air-conditioning system of a vehicle, and vehicle having such a ventilation/air-conditioning system (2015). URL `https://www.google.com/patents/WO2015007481A1`. WO Patent App. PCT/EP2014/063,320
52. Dykstra, R., Wayne, R.: Vehicle climate control method (2016). U.S. Patent, US9242531 B2